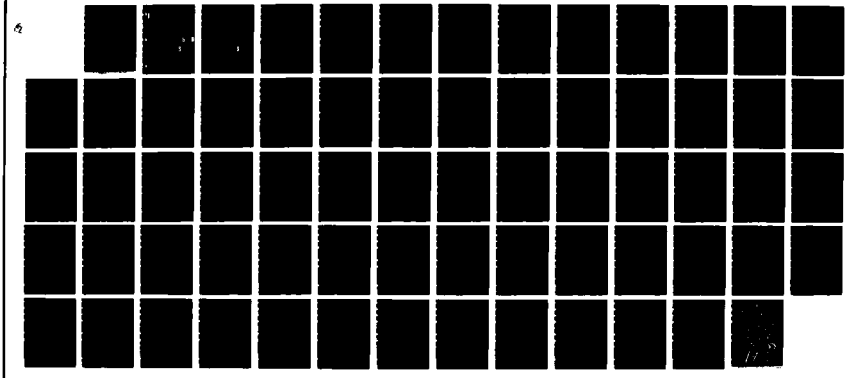
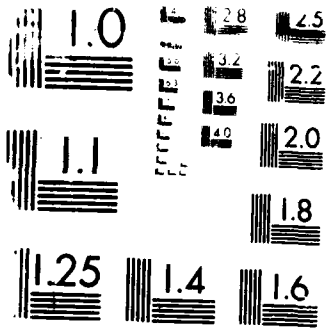
AD-A188 827 APPLICABILITY OF ADA (TRADEMARK) TASKING FOR AVIONICS EXECUTIVES(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGINEERING R E KONTAK NOV 87 1/1

UNCLASSIFIED AFIT/GCS/MA/87D-4 F/G 12/5 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

DTIC FILE COPY

AD-A188 827

DTIC
ELECTE
FEB 09 1988
S D
C&D

APPLICABILITY OF ADA TASKING FOR AVIONICS EXECUTIVES

THESIS

Roger E. Kontak
Major, USAF

AFIT/GCS/MA/87D-4

S E L E C T E

DISTRIBUTION STATEMENT A
Approved for public release
Distribution Unlimited

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

88 2 4 069

(1)

AFIT/GCS/MA/87D-4

APPLICABILITY OF ADA* TASKING FOR AVIONICS EXECUTIVES

THESIS

Roger E. Kontak
Major, USAF

AFIT/GCS/MA/87D-4

DTIC
ELECTE
FEB 0 9 1988
S
D

*Ada is a registered trademark of the United States Government (Ada Joint Program Office)

Approved for public release; distribution unlimited

AFIT/GE/MA/87D-4

APPLICABILITY OF ADA* TASKING FOR AVIONICS EXECUTIVES

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Information Systems

Roger E. Kontak, M.S.

Major, USAF

November 1987

Accesion For
NTIS CRA&I ☑
DTIC TAB ☐
Unannounced ☐
Justification
By
Distribution/
Availability Codes
Dist | Avail and/or Special
A-1

*Ada is a registered trademark of the United States Government (Ada Joint Program Office)

QUALITY INSPECTED 2

Approved for public release; distribution unlimited

Acknowledgments

I wish to thank a number of people who helped me during the preparation of this thesis. In particular, several members of the Aeronautical Systems Division Systems Engineering and Avionics Facility (SEAFAC) and the Air Force Wright Aeronautical Laboratory (AFWAL) provided essential assistance in the technical aspects of this thesis. Capt David King of SEAFAC gave me many hours of discussion on the design and implementation of avionics software and provided characteristic tasks to be performed by this software. He and SSgt James Bennett modified a JOVIAL version of an avionics scheduler which served as a baseline for my experiment. This version was more difficult to implement than initially envisioned and assistance was also obtained from 1Lt Stephen Ross, a former member of SEAFAC. 1Lt Ross also helped provide the inspiration for how to measure overhead in an avionics scheduler.

In addition to my sponsors at SEAFAC, several members of AFWAL provided both resources and technical assistance. I wish to thank Mr. Phillip Hanselman for several hours of discussion and allowing access to AFWAL owned computers and compilers. I also wish to thank 1Lt Marc Pitarys for his interest in my project and the time he spent helping debug the many problems encountered.

My AFIT classmates also provided encouragement and assistance. In particular, Capt Daniel Joyce and Cpt John Klemens (U.S. Army) helped during the many long hours and weekends we spent working together in the laboratory.

I also wish to thank my thesis committee. In particular, my thesis advisor, Capt David Umphress, provided encouragement and editing support throughout this document. The guidance and critical analysis obtained from my committee helped obtain coherent support for my thesis conclusions.

Finally, I wish to thank my wife and children for their patience, support, and understanding throughout this ordeal.

Roger E. Kontak

Table of Contents

List of Figures

List of Tables

Abstract

The purpose of this study was to evaluate Ada tasking performance and its suitability for avionics schedulers known as executives. This was done by comparing variations of Ada executives written by the author with the existing Digital Avionics Information System written in JOVIAL. The comparisons were made by evaluating the system overhead of each model while running a series of representative application tasks.

The study found that Ada tasking had considerably more overhead than its JOVIAL counterpart in order to maintain precise cyclical timing. Another outcome was that several Ada compilers were unable to produce code which could be run on the MIL-STD-1750A computer. This points to the present immaturity of Ada compilers targeted toward embedded aircraft computers.

Given the immaturity of Ada compilers, Ada tasking is not appropriate for avionics executives. Ada can still be used, however, without tasking and the associated Run Time System to develop executives.

This thesis adds support for the need to revise standards and develop compilers as necessary to provide an efficient Run Time System for Ada executives.

APPLICABILITY OF ADA TASKING FOR AVIONICS EXECUTIVES

I. Introduction and Problem Definition

Introduction

The Ada programming language was developed for the Department of Defense (DoD) in response to the perceived software crisis in the 1970's. A large portion of software costs related to this crisis are incurred for embedded computer systems. "By definition, an embedded computer system is one that forms a part of a larger system whose purpose is not primarily computational, such as a weapons system or a process controller" (Booch, 1983:13). For example, an aircraft embedded computer system may perform functions such as flight control, autopilot operation, weapons delivery applications, and similar routines. Embedded systems have particular programming requirements including:

-Parallel processing
-Real-time control
-Exception handling
-Unique input/output(I/O) control (Booch, 1983:13)

Ada was designed primarily to reduce embedded system software development costs (ARINC, 1986:1). Although real-time control is only one requirement for embedded systems, these systems are often referred to as real-time systems. A real-time system is one which must respond to externally generated input stimuli within a specified period of time. In other words, it has processing deadlines. "If a real-time

system performs the correct function, but delivers the result too late then it has failed to satisfy its requirements" (Auernheimer and Kemmerer, 1986:879).

The scheduler for real-time avionics systems that is responsible for delivering results on time is known as the executive. An executive which handles tasks at specific intervals is known as a cyclical executive. The executive provides an operating system for handling concurrent (parallel) processes which run on the embedded avionics computer. For Air Force embedded avionics systems, this executive has traditionally been written in the JOVIAL programming language. A JOVIAL executive, which usually contains assembly language subroutines, is created for each embedded system. This method, however, leads to program code segments that are not portable and may have software design inefficiencies.

Ada has a special construct, known as tasking, for writing executives. Tasks are entities that execute in parallel. Each task is considered to be executed by a logical processor of its own when run on a single processor system (DoD, 1983:9-1).

> These facilities are quite unlike the services provided by a typical run-time executive or operating system. Real-time systems will be designed as a set of cooperating concurrent processes (Ada tasks) using the Ada tasking model (Burger and Nielsen, 1987:49).

Executives written using Ada tasking promote the Ada goals of reliability and maintainability (Booch, 1983:47).

The Problem

Tasking is an important feature of Ada for real-time control of embedded systems. A programmer can use Ada tasking instead of JOVIAL to write the executive for a real-time system. Unfortunately, little is known about Ada tasking performance and its suitability for avionics executives.

Providing Ada tasking facilities creates execution overhead for the executive. This overhead must be better understood to determine if tasking offers a viable alternative to JOVIAL for writing avionics executives. To promote these goals, however, tasking must also operate efficiently without excessive overhead.

Scope

This thesis was designed to quantify Ada tasking performance in real-time avionics executives. The study was was directed toward evaluating executives run on the MIL-STD-1750A computer, a single processor embedded computer system used in Air Force avionics systems. The benchmarks developed for this computer should help programmers determine if Ada tasking can be used in place of JOVIAL in the avionics executive.

The intent of the thesis was to study tasking performance efficiency. Tasking errors and compiler problems discovered during the process have been identified, but investigation into correct tasking operation was not performed. For instance, no investigation was made into the

validity of shared variables among synchronized tasks.

This thesis compared Ada tasking performance in an avionics executive developed for this thesis to an existing executive written with JOVIAL. Since multiprocessor environments for Ada tasking are still under development and were not available for this study, only single processor implementations were evaluated. In addition, since bus I/O would be identical for Ada and JOVIAL, I/O operations were not utilized in either executive. The evaluation was made on a MIL-STD-1750A computer, the standard architecture for Air Force embedded applications, using program code compiled by Ada compilers targeted toward the 1750A. With this in mind, tasking performance was not investigated on other computer architectures. An attempt was made to evaluate whether compiler version significantly affected Ada tasking performance, but due to the immaturity of Ada compilers this factor could not be tested.

Thesis Overview

Chapter II of this thesis portrays pertinent findings from current literature. It begins with a look at high order languages and narrows the focus to Ada with emphasis on Ada tasking for avionics executives. The chapter also examines a JOVIAL executive used in this thesis. The chapter concludes with discussions of overhead determination and benchmarking.

Chapter III outlines the general experimental approach and requirements definition. It also describes the factors

which could have affected executive overhead. Finally, the methods of measuring overhead and evaluating this overhead with statistical tests are described toward the end of this chapter.

Chapter IV goes into greater detail of the experimental design by describing and comparing the Ada and JOVIAL executives. The specific statistical tools are also described in the latter part of this chapter.

Chapter V provides details of the experimental results and describes problems encountered with several Ada compilers. The results are analyzed to show the significance of each factor under consideration.

The results are then evaluated in Chapter VI. The evaluation leads to a conclusion of the applicability of Ada tasking for avionics executives.

II. Summary of Current Knowledge

Software engineering for avionics applications is an area of particular importance to the Air Force. "The real-time avionics environment differs from other environments that are supported by computer programs because of the timeliness required in responding to requests and because of the physical constraints of the hardware" (Witt, 1985:11). The hardware is constrained by size and weight limitations which require efficient utilization of available resources. In avionics, the designer does not have the luxury to add more memory when a means of operating with less memory is available. In addition, the speed required for certain avionics applications is very high. Although delay of a few seconds may be permissible with automatic radio tuning, delays of milliseconds in handling flight critical applications, such as engine controls, may be unacceptable (ARTEWG, 1986:17).

Ada is a standardized DoD programming language which is intended to reduce software life cycle costs by encouraging proper software engineering methodologies. These methodologies should result in:

-Increased understandability
-Increased reliability
-Increased modifiability
-Increased efficiency
-Increased reusability
-Reduced training needs
-Reduced schedule or technical risk (ARINC, 1986:2)

A recent DoD directive concerning the use of Ada was

instituted with these benefits in mind. DoD directive 3405.1, signed April 2, 1987, states:

> The Ada programming language shall be the single, common, computer programming language for Defense computer resources used in intelligence systems, for the command and control of military forces, or as an integral part of a weapon system.
> (DoD, 1987:1)

This directive, while mandating the use of Ada, does not specifically require the use of tasking for weapon systems. Tasking will be used only if it operates efficiently enough for avionics systems.

To evaluate Ada tasking, this literature review first investigates current material on tasking's applicability, and then lays the basis for the experimental portion of this thesis. It begins with a look at why High Order Languages (HOL) are necessary for avionics executives. This viewpoint substantiates the rationale for using the Ada programming language and then expands in the next section to consider the applicability of Ada tasking. Considerable controversy exists concerning Ada tasking feasibility for avionics, so pertinent issues are outlined in the third section. These issues are not directly addressed in this thesis, but point to some tasking problems which remain unsolved. The problem addressed by this thesis concerns tasking performance efficiency which is specifically addressed in the next section on overhead determination. Finally, the last section on benchmarking lays the groundwork for benchmarking techniques which are used in this thesis.

High Order Languages

In order to study Ada tasking, the overall philosophy of using HOL's for avionics executives must be examined. This review is necessary because software written with a HOL can be accomplished as efficiently or more efficiently with assembly language (Rubey, 1978:947). Fisher (1980:1) notes that some believe HOLs should not be used because:

-HOLs produce time inefficient machine code
-Resulting HOL-generated code uses too much memory
-HOLs lack the syntactic structures for real-time programming

In spite of the assembly proponents such as Rubey and Fisher, there is considerable debate concerning applying HOLs to avionics executives and other real-time systems. The primary argument for use of HOL's is that "software development and maintenance costs will make the use of HOL cost-effective in spite of some loss of memory and execution efficiency" (Lindley, 1980:1).

Hardware and software trends are solidifying the argument for using HOL's in embedded systems. The main factors supporting the use of HOL's include rising labor costs, increasing software complexity, declining hardware costs, and reductions in hardware size and weight for given functions (Lindley, 1980:1). While avionics computer hardware is becoming less expensive and more compact, the corresponding software is becoming larger and more complex. Software is becoming more difficult to develop, debug, and maintain (Lindley, 1980:4). Simultaneously, hardware trends

are causing a decrease in factors which favor assembly language for extracting the greatest possible performance out of a particular processor and memory space (Lindley, 1980:5).

The benefits of using an HOL become clear when considering life cycle costs. HOL programmer productivity is at least doubled since each line of HOL code corresponds to several lines of assembly language code (Lindley, 1980:22). In addition, software management tools are more easily applied to HOL based projects (Lindley, 1980:23). Later in the life cycle, software maintenance costs are lower because changes can be made without adversely affecting the program's overall unity. Similarly, maintenance is also easier because HOL programs are inherently more readable (Lindley, 1980:14).

To take advantage of HOL life cycle efficiencies, the Air Force has primarily used the JOVIAL HOL for avionics executives. A study which compared the Digital Avionics Information System (DAIS)[1] written in JOVIAL to an assembly language version supports the use of HOLs (Trainor, 1967). Although an approximate ten percent degradation was incurred for both memory and execution time in the JOVIAL version, programmer productivity was more than double that of the assembly version (Trainor, 1976:6,7). Furthermore, the JOVIAL implementation was much easier to read and interpret,

[1]The DAIS is a standardized architecture for avionics systems designed to accommodate a wide variety of avionics configurations. The DAIS architecture includes a software executive and application software. The DAIS will be covered in greater detail in chapter 3.

making it more reliable and maintainable than its assembly language counterpart.

Since Ada was specifically designed with embedded systems in mind, it may further reduce software life cycle costs. To exhibit the benefits of Ada, a study was performed in which the DAIS was recoded from JOVIAL into Ada (Scarpelli, 1980:5). Although the Ada version was not fully implemented (no compiler was available at the time), Ada displayed several advantages over JOVIAL. First of all, Ada promoted top-down structured programming. Additionally, the Ada source code was self-documenting and easier to read. Another advantage was that transporting programs from one machine to another would be easier.

Finally, strong typing provided the advantage of reducing subtle type errors and maintaining data integrity. Several disadvantages were also expressed as a result of the study. These include the necessity of assembly language to perform certain functions as well as the amount of data manipulation required to change data format. The biggest concerns of using Ada are the amounts of memory and run-time overhead (Scarpelli, 1980:31).

A similar study known as Tactical Ada Guidance (TAG), performed by the Air Force Armament Laboratory at Eglin AFB, highlights these concerns in an investigation of Ada's applicability to operational flight software for the Medium Range Air-to-Surface Missile (MRASM) (Schnelker et al., 1985). The original JOVIAL implementation for this missile was

redesigned, coded, and tested using Ada. As seen from Tables I and II, the design and coding phases for each version took approximately the same amount of time. The increased times for the Ada version were due largely to inexperience with Ada (Schnelker et al., 1985:vi). More pronounced increases are evident in the code expansion and CPU utilization of the Ada version. Although Schnelker et al. (1985:vi) note that an optimized compiler should bring these figures down to acceptable levels, the memory consumption and run-time overhead substantiate the concerns raised by Scarpelli (1980:31).

Table I. TAG Design Phase

	TAG (Ada)	MRASM (JOVIAL)
Design phase in man-months	12	10
Lines of PDL	1030	900
Amplifying comments	40	200
Number of tasks	16	8
Number of functions/procedures	32	30

Table II. TAG Coding Phase

	TAG (Ada)	MRASM (JOVIAL)
Coding phase in man-hours	720	500
Lines of code	3034	3212
Amplifying comments	173	172
Number of tasks	17	10
Number of functions/procedures	25	26
Code expansion (bytes-dec)	28860	9292
Duty cycle (% CPU utilization)	52%	13%

(Schnelker et al., 1985:vi,vii)

Ada Tasking

While the Ada version of the DAIS did not use tasking, the laboratory which performed the comparison believed the executive could have been designed and coded more efficiently

using the Ada tasking features (Scarpelli, 1980:31). Since the time of the study, the Air Force Wright Aeronautical Laboratories have extended their research to include Ada tasking. A current project, the Common Signal Processor, evaluated Ada tasking for use in a distributed computer system. Tests early in the program, however, found Ada tasking to be too inefficient in terms of execution speed (Hanselman, 1987). Similarly, Ada tasking was found to be sufficient for the TAG project, but Schnelker et al. (1985:63) felt that Ada was not appropriate for time-critical software which interacts with sensor devices due to the simplicity of the Ada tasking model which requires more tasks for an embedded multi-task system than a custom designed operating system (Schnelker et al., 1985:viii).

Tasking also has memory inefficiencies, since additional storage is required to keep track of a task's state, the status of delays, and a list of tasks waiting on entry calls (Baker and Riccardi, 1985:36-38). Problems such as memory inefficiencies have raised serious doubts whether Ada can be used effectively in embedded system applications. A project undertaken by the Computer Sciences Corporation yielded the following conclusion:

> We believe that the Ada language tasking model, including the preemption/priority scheme and the termination mechanism, may not be sufficient to support a typical real-time tactical embedded computer system. We are concerned too, [sic] about the efficiency which the Ada language run-time systems will support recoded executive functions and about the portability of such systems.
> (Friedman, 1987:176)

Due to the doubts concerning Ada for avionics applications, Ada programmers often resist incorporating its new features in software design methodology. It is possible, for instance, to write an executive in Ada without using tasking constructs (Phillips and Stevenson, 1984:103). This method of programming defeats much of the purpose of using Ada to promote program modularity and portability.

A great deal of the reluctance to use Ada or its tasking construct is a result of the uncertainty about performance in embedded avionics systems (Phillips and Stevenson, 1984:103). The problems associated with tasking overhead are amplified by Phillips and Stevenson (1984:103).

> The Ada scheduler for a general system is designed for asynchronous tasks unique to this machine and compiler implementation, and may provide an unknown and potentially disastrous amount of overhead for tasks that have strict time limits on the total amount of time they can take to execute. As a result of this, real time systems would probably tend to be written in such a way that avoids invoking any scheduling actions. This means that Ada's very nice asynchronous event handling properties may not be taken advantage of when the cyclical executive is implemented directly by the user.

Before evaluating tasking overhead in greater depth, the advantages and disadvantages associated with Ada tasking must be considered. A study of tasking by the Mitre Corporation (Carrington, 1986) identified the advantages and disadvantages shown in Figure 1.

High overhead occurs in several instances within Ada tasking. During a rendezvous, which involves coordination between two tasks, context switching from one task to another

Advantages:

-Tasks provide a convenient means for designing concurrent processes and mapping these processes to the target architecture.

-Tasks provide a means of task communication directly without communicating through databases.

-Priorities allow control over relative task execution urgency.

-The delay statement allows control over task timing.

-The run-time environment handles the details of task synchronization, communication, and context switching.

-Ada is capable of handling interrupts (an essential feature of event-driven, real-time applications).

-Leaving task activation and termination to the run-time environment reduces the coding and testing effort.

Disadvantages:

-Lack of a dynamic priority mechanism.

-No means to explicitly control task activation.

-No means to temporarily suspend a task.

-Due to a lack of centralized tasking control, Ada requires a high level of coupling among programmers.

-No explicit cyclical executive facility exists.

-Vendor interpretable definition of delay statement.

-High overhead associated with the rendezvous and implementation dependencies.

-Difference in interrupt handling facilities from one Ada implementation to another limit efforts to develop reuseable software.

-High overhead associated with elaboration of tasks which must be activated in a particular order.

Figure 1. Ada Tasking Advantages and Disadvantages (Carrington, 1986)

can be intolerably slow for embedded systems (Kamrad, 1984:473). Likewise, the unpredictable overhead incurred during task creation and termination makes dynamic task allocation inappropriate for avionics. Therefore, no tasks should be created or terminated during normal program operation (ARINC, 1986:17).

Another feature of Ada tasking which complicates executive programming is that the semantics of Ada are inherently asynchronous (Adams, 1983:982). Thus, event-driven asynchronous processing is easily accommodated, but the traditional cyclical executive does not take advantage of Ada's strengths (Softech, 1986:14-7). A pure cyclical executive has the advantages of familiarity, simplicity of runtime system, efficiency (low overhead), and predictability. On the other hand, asynchronous processing has the advantages of more natural partition of problem, flexibility, and ease of design (Softech, 1986:14-18). In typical real-time applications, certain actions such as data sampling and control loops must be performed repetitively (Softech, 1986:14-1). This is best done by the cyclical executive to prevent tasks from falling behind and ultimately causing control loops to get out of phase with the rest of the system (Softech, 1986:14-10).

Another problem associated with the asynchronous nature of Ada tasking is that it does not guarantee fairness. Thus, a task may be permanently blocked waiting for a rendezvous (Mundie, 1986:24).

Due to these problems, the popular approach to the cyclical executive has been to bypass Ada tasking constructs by using an interrupt-driven executive (Phillips, 1984:4.103). A proposed method of implementing a cyclical executive in Ada is to develop a pragma to supply information not accessible to the compiler (Phillips, 1984:4.104,105). The effect of the pragma is the same as associating an interrupt with a task entry at each desired frequency (Phillips, 1984:4.105).

While Ada is to be used in all future avionics systems, a certain amount of assembly language programming can be expected in many systems. This is because certain functions are more critical to the efficiency of the system since they are executed at a high frequency. Coding these critical functions in assembly language will improve overall system efficiency. In fact, coding approximately five to twenty percent of a program in assembly language can produce a program which executes nearly as efficiently as a program written in assembly language (Lindley, 1980:16).

Overhead Determination

A recent study by Burger and Nielsen (1987:49) of Hughes Aircraft Company measured task overhead in a similar manner specifically for DEC Ada (version 1.2) on a VAX 8600. A summary of their findings, giving the amount of overhead in microseconds and normalized relative to the cost of a procedure call, is provided in Table III.

Table III. Tasking and Exception Handling Overhead

	Description	Overhead usec	Overhead normalized
1.	Task activation and termination	1960	178
2.	Task created via an allocator	150	14
3.	Procedure call	11	1
4.	Producer-Consumer (2 context switches)	503	46
5.	Producer-Buffer-Consumer	1220	111
6.	Producer-Buffer-Transporter-Consumer	1694	154
7.	Producer-Transporter-Buffer-Transporter-Consumer	2248	204
8.	Relay	906	86
9.	Conditional Entry		
	- no rendezvous	170	15
	- with rendezvous	29	3
10.	Timed Entry		
	- no rendezvous	254	23
	- with rendezvous	33	3
11.	Selective Wait with Terminate	127	12
12.	Exception in a block	222	20
13.	Exception in a procedure	217	20
14.	Exception during a rendezvous	962	87

(Burger and Nielsen, 1987:56)

Burger and Nielsen conclude, from the high cost for activation and termination of tasks, that " . . . tasks for time-critical modules should be declared within packages or in the main procedure, rather than in procedures or tasks within the system" (Burger and Nielsen, 1987:57). An additional conclusion is that the overhead for a rendezvous is indicated by the producer-consumer relation '500 used'. This overhead is expected to occur every time two tasks are in a rendezvous and does not include any execution time for statements within the body once a rendezvous occurred (Burger and Nielsen, 1987:57).

Conditional and timed entry call have higher overhead when a rendezvous does not occur than when one does take place. This overhead should be considered when polling is used to establish synchronization between two tasks. An additional consideration is to avoid placing a selective wait with terminate option inside a loop since its overhead is incurred each time the loop is executed (Burger and Nielsen, 1987:57). Burger and Nielsen emphasize that these findings are based upon tests with DEC Ada on the VAX 8600 and projections cannot be made for execution of the same benchmarks on other architectures or other compilers.

Benchmarking

Using benchmarks such as those created by Burger and Nielsen involves a number of complex operations. These include:

-Isolating the feature to be measured;

-Achieving measurement accuracy and repeatability;

-Eliminating underlying operating-system interference from time slicing, daemons, and paging.

(Clapp et al., 1986:760)

Another problem found when comparing two versions of a program, such as HOL versus assembly language version, is that writing a complete baseline assembly language version would more than double software development costs (Rubey, 1978:947).

To solve the problem of versions for comparison purposes, only a few segments of a complete program are used for comparison. This type of comparison raises the additional question of whether the segments are representative of the entire program. There is additional uncertainty as to whether the baseline version could be made smaller or faster by a better programmer. Usually, an assumption must be made that the baseline version contains representative segments that are the product of an average programmer (Rubey, 1978:947).

The most commonly used technique for measuring the time needed to perform an operation is to execute the operation a large number of times and take time readings only at the beginning and end. The desired time is then found by averaging (Clapp et al., 1986:762). This technique still leaves the previously identified complex operations to deal with. To isolate a feature to be measured, the control and

the test segment must differ only by the feature being measured. In spite of controlling differences in corresponding tests, code optimization can distort benchmark results and must be avoided (Clapp et al., 1986:762). Measurement accuracy is achieved by statistically determining the number of iterations needed to obtain a parameter measurement within a given tolerance (Clapp et al., 1986:762-763). Finally, eliminating the underlying operating system is done by running the tests with no other user processes in concurrent execution and all daemon processes disabled. Even with these precautions there are still timing anomalies that must be measured and detected (Clapp et al., 1986:764).

Conclusion

Although the use of Ada has been mandated for use in all future weapon system development, this mandate does not imply that tasking be used. Preliminary studies of tasking for real-time systems have identified several shortcomings primarily concerned with performance. These studies have created uncertainty for the future of Ada tasking as it presently stands. Further study of tasking efficiency is needed to determine if tasking provides a viable method to structure avionics executives. If so, tasking uncertainties must be exposed to promote using tasking in future systems.

III. Experimental Design

General Approach

The main goal of this thesis was to investigate the overhead associated with tasking for avionics executives. This goal was pursued by comparing two executives: one using Ada tasking and the other using task scheduling written with JOVIAL J73/I.

The rationale for comparing different executive models was twofold. First, empirical measurements of overhead for individual Ada tasking features alone have already been made in studies by Burger and Nielsen (1987) and Clapp et al. (1986). Second, the suitability of Ada tasking for avionics systems must be evaluated with respect to overall system performance. Since JOVIAL J73/I is the previous standard for Air Force avionics software, comparing an Ada version with a JOVIAL implementation of the same system yields an analysis of how well Ada tasking compares with the previous standard.

The first step of performing an experiment to compare Ada and JOVIAL was to determine requirements for the experimental design. Rather than attempting to formulate these requirements, a search was made for existing JOVIAL executives which could be rewritten using Ada tasking. To be useful, the chosen JOVIAL executive had to be representative of avionics systems. The only executive found to be available for this study was a subset of the DAIS

executive. Fortunately, the DAIS is highly representative of avionics executives and variations of the DAIS are actually in use in many avionics applications (King, 1987). The DAIS is an ideal executive to study since it continues the research done by Scarpelli (1980) and also expands work done by SEAFAC which used the DAIS for a KC-135 executive.

> DAIS is a system architecture which can be configured for various avionic [sic] applications and missions using core elements or building blocks. The purpose of the DAIS concept is to reduce the proliferation and nonstandardization of aircraft avionics, and permit the Air Force to assume initiative in the specification of standard avionic [sic] systems and interfaces for future Air Force system acquisitions. (DAIS, 1977:5)

To provide flexibility for various avionics configurations, the DAIS is driven by tables which contain lists of all application tasks and specific task requirements. The tables to run the DAIS were built using a scaled down, yet representative, series of application tasks for a typical avionics executive. The representative tasks were provided by SEAFAC based upon their experience with avionics executives. The task specifications included the phase, frequency, and priority required for task execution. For a more detailed explanation of these specifications and the detailed design of the experiment; see Chapter IV. Since the tests were designed to study the task scheduling overhead of each model, the function of each application task was immaterial to the experimental design. Thus, each application task contained the same body. The task bodies were designed to work within the Ada executive as well as the

DAIS. Rather than use null task bodies, the task bodies were designed to prevent task elimination through Ada compiler optimization by using global variables which are referenced both inside and outside the task bodies.

After running the DAIS on a MIL-STD-1750A computer, an Ada tasking executive was designed and written to schedule identical application tasks. This executive was originally debugged, run, and tested on a DEC VAX-11/782[2]. After program errors were eliminated, the actual tests and comparisons of both version were performed on the 1750A computer. Since the 1750A nas no operating system to assist with performance measurement, running the executives on the 1750A presented more difficulty in obtaining measurements. The measurements obtained, however, were more accurate because there was no interference from the operating system nor multiple users to affect the experimental outcome.

Measures and Factors

The primary performance concern of an avionics executive is run-time processing efficiency (Dewar, 1987). To measure overall run-time efficiency, the run-time overhead must be measured. Ideally, the run-time overhead should leave enough CPU time available to give application tasks primary access to the processor. To measure run-time overhead, idle CPU time was recorded and compared for each model. Since each model performed identical application tasks, the

[2]The DEC VAX-11/782 is an uncommon computer which uses two VAX-11/780 processors.

difference in idle CPU time indicated the model's relative run-time efficiency. Several tests were subsequently developed to determine whether the Ada tasking version significantly affected task execution.

To make this determination, all factors which could affect task execution were considered and included in the experiment as necessary. The factors which could affect this performance are illustrated in Figure 2.

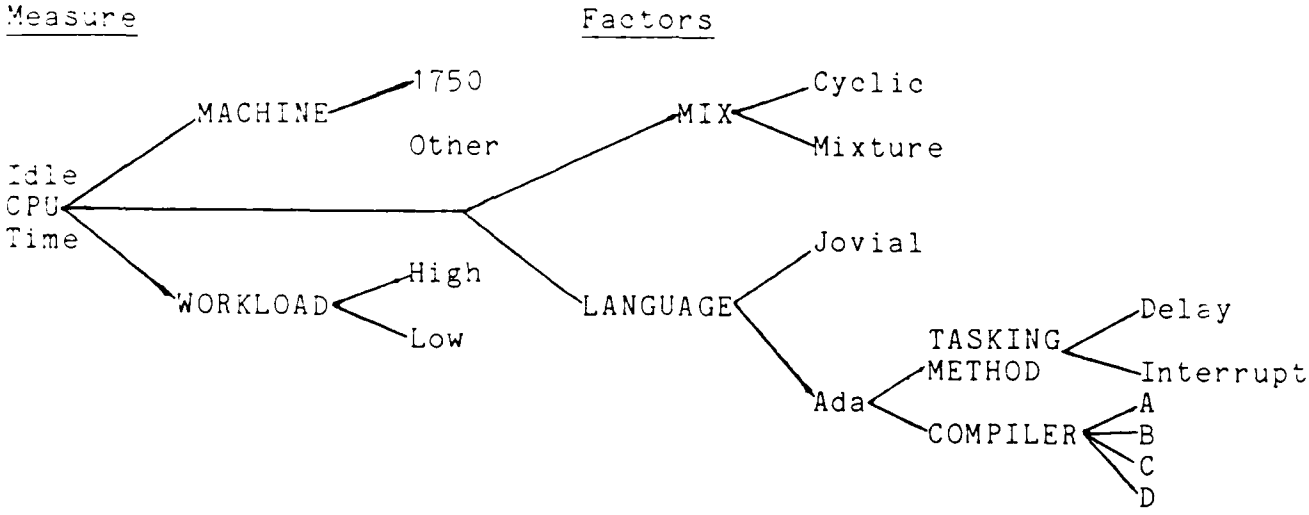


Figure 2. Experimental Factors

Machine.

Although the hardware architecture and specific machine features could have an effect on performance, this factor was not investigated. The 1750A architecture alone was used since it is the standard architecture for embedded systems. Using the 1750A also minimized measurement errors since it is a single user machine with no time sharing and no operating system to influence results. To avoid hardware interaction,

all tests were performed on the same machine.

Mix.

Avionics executives typically consist of a mix of cyclical and asynchronous tasks. Cyclical tasks are necessary for many avionics tasks that must be repeated at a specific frequency such as control (feedback) loops or data sampling (King, 1987). In addition, cyclical tasks tend to spread out processing demands to avoid having several tasks waiting to run simultaneously. Asynchronous tasks, on the other hand, are triggered by events which are not predetermined, and perform functions peripheral to cyclic tasks. Ada tasking is geared more toward handling asynchronous tasks. To study the effect of asynchronous versus cyclical tasks, the experiment first looked at purely cyclical scheduling and then determined if adding asynchronous tasks at random intervals affected performance in either JOVIAL or Ada environments. Solely asynchronous tasks were not evaluated since they would not represent a typical avionics executive (King, 1987). Asynchronous tasks were instead combined with cyclical tasks to utilize both methods of task scheduling and to increase the tasking workload.

Workload.

Another method used to study the effect of workload was to shorten the major frame. The workload was studied in this manner to analyze the effect of workload alone. A major frame is the longest period of time specified for synchronous

action. In other words, it is the time interval of the least frequently occurring cyclical task. By reducing the major frame and thereby increasing the frequency of all cyclical tasks by a corresponding amount, the workload was increased. Although workload is a continuum, it was studied only at two levels to facilitate statistical analysis as described in the next chapter.

Language.

The JOVIAL J73/I versus MIL-STD-1815A Ada language performance was the primary factor under investigation. Since these languages are standardized language variations were not considered.

Compilers.

In spite of language standardization, the compiler used was expected to have a significant impact. Since JOVIAL is a mature language, only the Air Force standard JOVIAL compiler was used. Compiler version was expected to have a greater affect on the Ada version, however, so the four 1750A Ada compilers available for this thesis were tested with identical code. Of the four compilers, three were unable to compile and run the code successfully. The problems encountered are discussed in Chapter V.

System Timing Method.

In addition to the compiler effect, the method of implementing Ada tasking for cyclical tasks was considered. One method, that of using delay statements, uses pure Ada tasking features to achieve cyclical scheduling. Since

delays are the minimum time period a task must wait, a task sometimes resumes execution a short time after its delay expires. As seen in Figure 3, the task gets out of phase with the rest of the system, and ultimately operates at less than the required frequency. This phenomenon is termed cumulative drift (Softech, 1986:14-10).

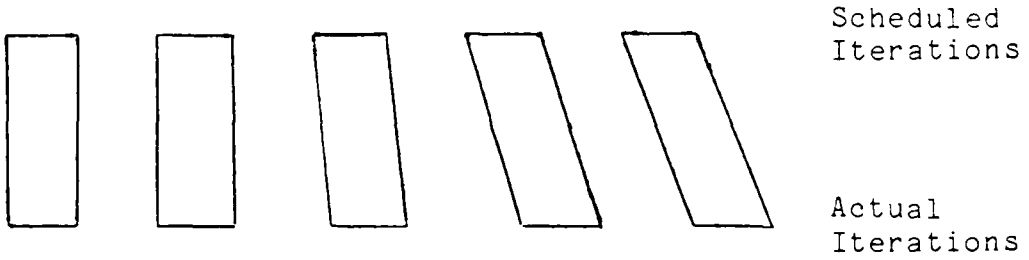


Figure 3. Cumulative Drift
(Softech, 1986:14-7)

An alternative method which avoids cumulative drift by associating task entry calls to timer interrupts was tested and evaluated. Although other methods of dealing with cumulative drift exist, this is the simplest method and represents the least amount of overhead. Some perceive, however, that this method avoids proper tasking constructs and does not take advantage of Ada's strengths (Softech, 1986:14-7).

Statistical Tests

The factors under investigation were analyzed in accordance with procedures for factorial designs at two levels. These tests looked at two levels for each factor under consideration. For instance, when considering the

workload factor, high workload and low workload comprised the two levels. All combinations of factors must be tested unless a fractional factorial design is used. In this thesis, all combinations were tested except those cases which would not run due to compiler limitations. Further details of the statistical tests are provided in Chapter IV, and test results are found in Chapter V.

Conclusion

The measurement used to determine the applicability of Ada tasking for avionics executives was the amount of idle CPU time in a major frame. Greater CPU idle time indicated less spent in tasking overhead. Since both versions performed the same tasks, the comparison showed the difference in overhead for each combination of factors. To determine which factors affected tasking performance, the factors shown in Figure 4 were considered.

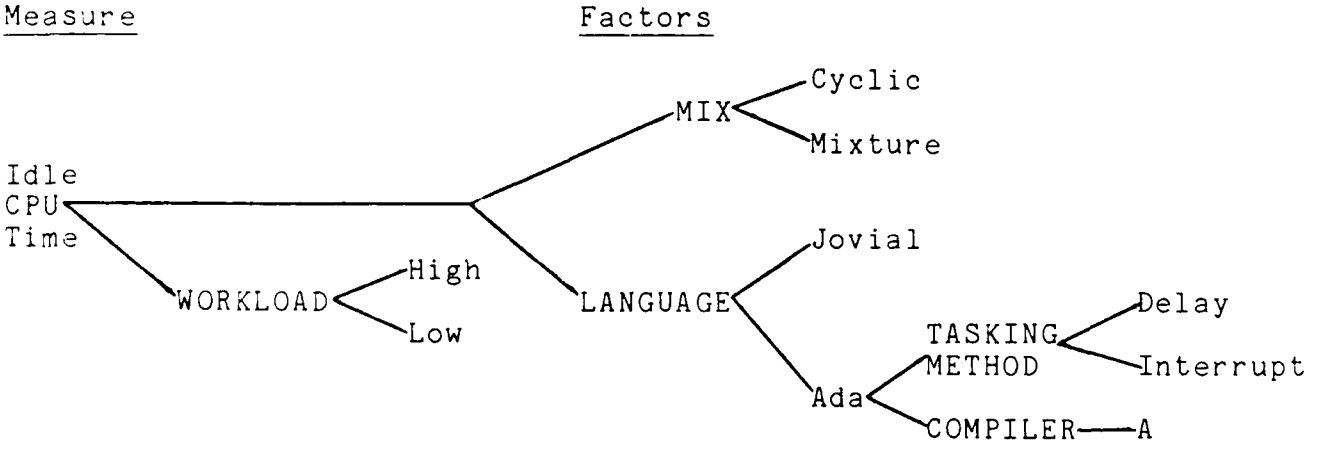


Figure 4. Tasking Performance Factors

IV. Detailed Design

Although the Ada tasking executives and the JOVIAL DAIS executive were designed to handle tasks similarly, the Ada models used the Ada Run Time System (RTS) to handle task interaction while the DAIS contained its own task scheduling functions. This chapter begins by explaining design details which are common to all models and then studies each model separately. A comparison of the models reveals a great degree of conceptual similarity. The implementation differences account for different amounts of overhead for each model. The chapter ends with a description of the statistical methods used in comparing executives and analyzing results.

All models are based on a real-time system in which tasks are coordinated with the passage of time. The minimum time granularity in which task activation can be specified to occur is known as a minor cycle. A major frame, the longest period of time which may be specified for a synchronous action to occur, was defined as 128 minor cycles for each model. Initially, both versions were run with a four second major frame duration yielding minor cycles of 1/32 of a second. This time is typical of avionics executives and was subsequently reduced in each model to study the effects of increased workload.

All models were tested with the same series of

applications tasks so that comparing executives would show differences in overhead. The cyclical tasks used for these tests are shown in Table IV. These tasks were chosen to realistically model the cyclical tasks found in a typical avionics executive (King, 1987). The phase of a task, which indicates the initial minor cycle for a given task, prevented having too many tasks attempting to run in the same minor cycle. The phase distributed the demand on processing resources evenly over all minor cycles.

Table IV. Cyclical Tasks

Period	Tasks	Priority (1-low)	Phase
2	A2	2	1
4	A4	3	3
8	A8	4	7
	B8	4	6
16	A16	5	15
	B16	5	14
32	A32	6	31
64	A64	7	63
128	A128	8	0

Asynchronous tasks were subsequently added to each model by calling a linear congruential pseudo-random number generator during each minor cycle. Random numbers were used to represent random events which trigger asynchronous tasks. Depending on the random number encountered, each minor cycle would contain zero to nine asynchronous tasks.

The correspondence of random numbers with a particular number of asynchronous tasks was chosen to realistically model the asynchronous task workload in a typical avionics executive (King, 1987). As shown by the figures displayed in Table V, there is a high probability of few asynchronous tasks in a given minor cycle and a low probability of all nine asynchronous tasks occurring in one minor cycle.

Table V. Asynchronous Task Activation

Random Number Generated	Number of Asynchronous Tasks called
1	9
2 .. 3	8
4 .. 6	7
7 .. 10	6
11 .. 17	5
18 .. 26	4
27 .. 37	3
38 .. 49	2
50 .. 65	1
66 .. 100	0

Workload, the last factor common to all models, was studied by changing the time between minor cycles. A shorter time between minor cycles represents a higher workload per period of time. The remaining factors, Ada compiler selection and system timing method, were compared only with each Ada version since these factors do not apply to the DAIS model.

Having explained design details common to both models, the DAIS and Ada models will now be described separately in greater detail.

DAIS Model[3]

The DAIS uses events for task scheduling purposes. Events refer to occurrences such as the start of a new minor cycle or actuation of a switch which could trigger task activation. Random events such as actuation of a switch are simulated by random numbers in this experiment. System timing to establish the interval between minor cycles is performed by the master executive which sets a hardware clock to interrupt whenever one minor cycle has elapsed (ASD, 1980:47). The DAIS controls task states by referencing the table of tasks and scheduling tasks with respect to events and priority. The task states, along with the method of transition from one state to another are shown in Figure 5.

When the DAIS is started, all tasks are in the invoked state. From this state, events occur (such as a new minor cycle) to make a task active. The active task is then performed according to processor availability.

[3]The DAIS source code is available from SEAFAC (ASD/ENASF) WPAFB, OH.

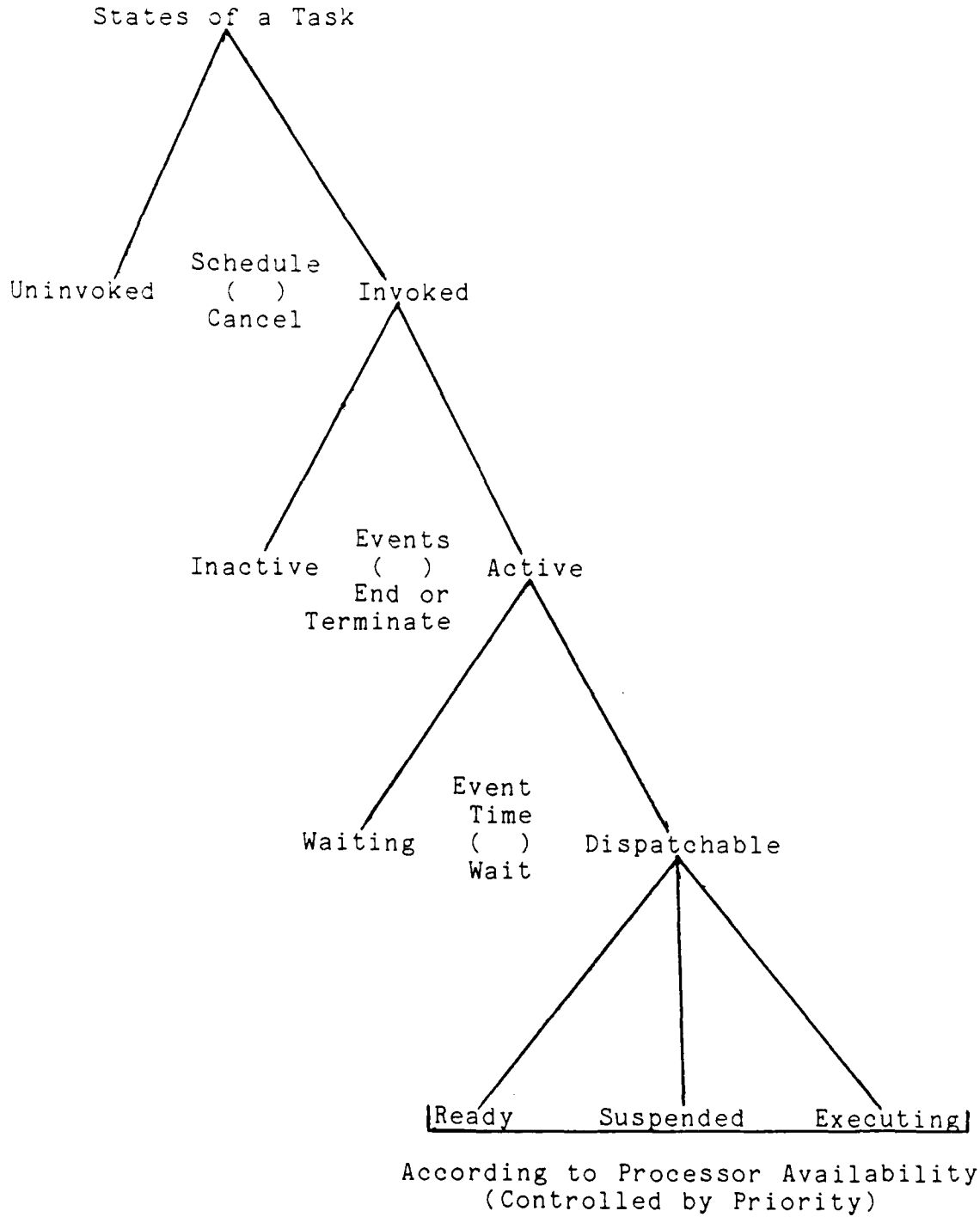


Figure 5. DAIS Task States
(ASD, 1980:10)

Ada Tasking Model

The Ada tasking model uses the Ada RTS to control task state in a manner similar to the DAIS executive. The possible states of an Ada task are shown in Figure 6. Although the Ada task state diagram is simpler than the DAIS diagram, a similarity appears in possible task states. The similarity is shown in Table VI.

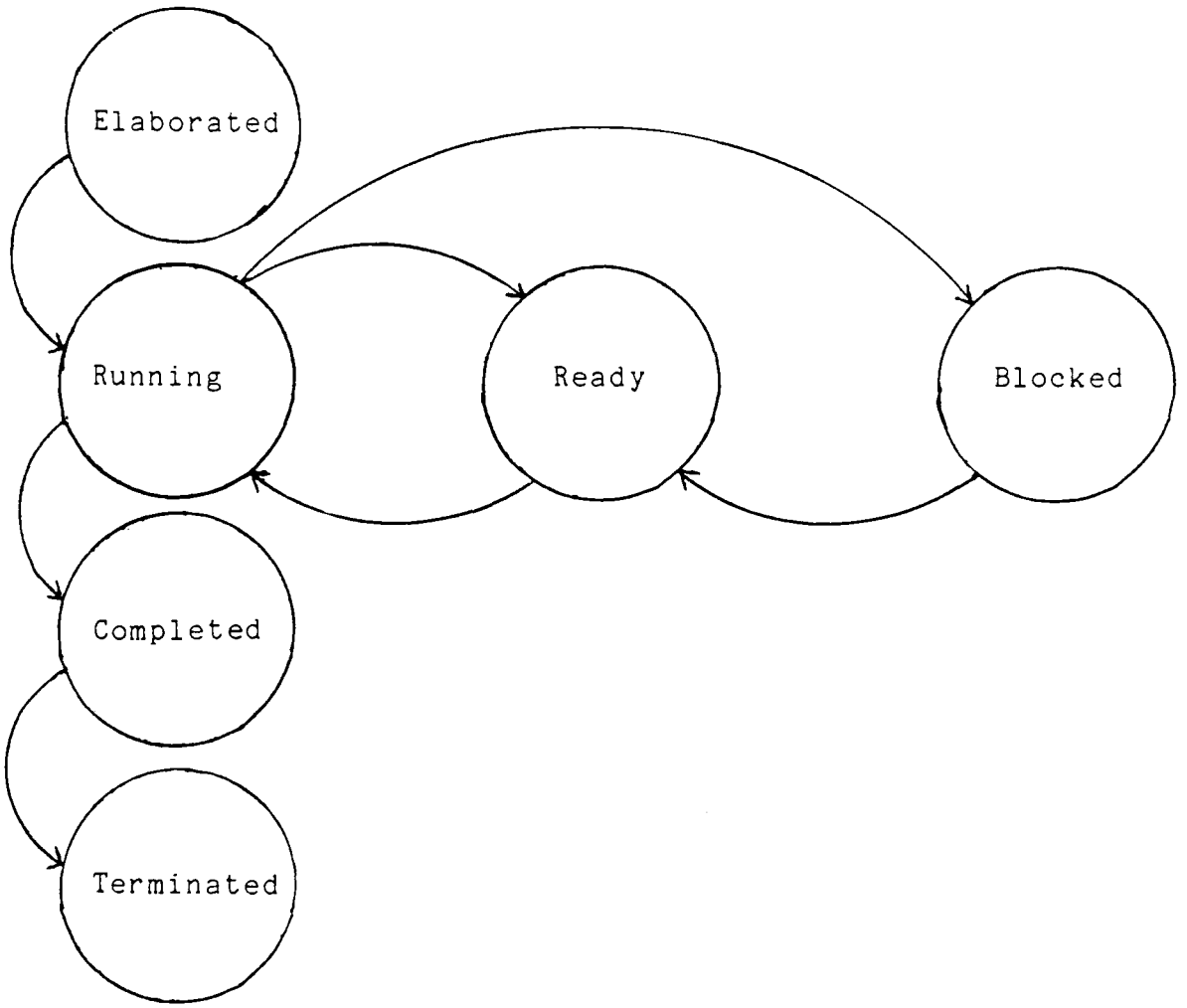


Figure 6. Ada Task States
(Booch, 1986:282)

Table VI. Corresponding States

Ada	DAIS
Elaborated	Invoked
Running	Invoked-Active-Dispatchable-Executing
Ready	Invoked-Active-Waiting
Blocked	Invoked-Active-Dispatchable-Ready
Completed	Invoked-Active-Waiting
Terminated	Uninvoked

System timing in an Ada executive is not handled automatically by the RTS but must instead be established by the program designer. The delay construct can be used to establish the desired interval between minor cycles. Another alternative is to set a hardware clock, as in the DAIS, and tie the clock interrupt to the start of a new minor cycle. Both methods of system timing were evaluated in this experiment to determine the significance of tasking methodology on system overhead as will be explained in the next sections.

Ada with Delay Statements.[4]

Cyclical tasking is difficult to implement in Ada without using interrupts because the RTS provides no automatic means of system timing. Nevertheless, it can be done with a minor cycle task calling a scheduler task at the beginning of each minor cycle. The scheduler task in turn calls each task to be run during the minor cycle. The program structure is shown in Figure 7.

Tasking entry calls were used to synchronize between the

[4]The Ada source code is available from AFIT/ENC WPAFB, OH.

Minor_Cycle and Scheduler tasks and between the Scheduler and application tasks. Asynchronous tasks were subsequently added to this model by calling the random number generator from within the Scheduler task. The workload was changed by changing the minor cycle duration.

```
task Minor_Cycle
  loop 1 .. Number_of_Major_Frames
    loop 1..128
      start Scheduler
      delay Minor_Cycle_Duration
    end loop
  end loop
end Minor_Cycle


task Scheduler
  loop
     accept start do

    If proper period, phase and conditions then
      start appropriate application task
  end loop
end Scheduler

task cyclical_task
 loop
    accept start do
      counter := counter + 1
      if counter > 5000 then
         counter = 0
 end loop
end A2
```

Figure 7. Ada Program Structure

Ada with Interrupts.

In an attempt to obtain more accurate system timing, a similar tasking model was developed with minor cycle timing handled by an interrupt tied to a hardware clock. The interrupt was used to call the minor cycle task rather than

using a delay to maintain system timing.

The clock is set and the interrupt vector is handled through 1750A assembly language programs. These programs are combined with the Ada programs and called through Ada's pragma interface. A separate assembly routine is needed for each minor cycle duration desired.

This model was also tested with the same asynchronous tasks and by varying the workload. The results of all test combinations are found in Chapter V.

Overhead Measurement and Statistical Analysis

Overhead Measurement.

Determining how to measure executive overhead was the most pervasive problem of this thesis. Executive overhead could not be isolated for measurement with calls to a system timer. The lack of operating system tools in the Ada RTS to measure processor utilization also hindered timing measurements. The only method found to measure overhead was to measure idle CPU time.

Idle CPU time was tracked by a low priority task which incremented a value whenever higher priority tasks were not ready to run. The count was performed through an assembly routine which incremented a register every time called. Overflows were detected and saved in a second register. At the end of each major frame, these registers were read and reset back to zero. The CPU idle time was then analyzed by comparing the number of increments.

In the Ada models, the assembly routine was called by a single task of low priority which executed in an infinite loop. Since the DAIS would not allow a task to run in two consecutive minor cycles, two separate low priority tasks were established. One task called the assembly routine in phase zero and the other called it in phase one. Both tasks had a period of two so they would execute every other minor cycle.

Statistical Analysis.

The first factors tested were those pertinent to the Ada models alone. Unfortunately, only one compiler was able to successfully compile and run the programs, so the compiler effect could not be evaluated. The effect of the system timing method was analyzed, however, and used to determine the significance of this factor. Once this factor was evaluated, all combinations of remaining factors were compared between the DAIS, Ada with delay statements, and Ada with interrupts.

The first step of the statistical analysis was to analyze the variability of results to determine how many repititions of the experiment were necessary. Each run consisted of measuring the idle time in one major frame. The first major frame was excluded since all tasks were being initialized. Subsequent major frames gave consistent results. The number of major frame samples required was determined by first taking five samples. The number five was chosen arbitrarily. According to statistical procedures, the

initial number of samples is unimportant and only the judgment of the experimenter is used to derive this number. Four samples (or nearly any other number of samples) would have worked just as well, but for this thesis five were used. These samples yielded an estimate of the variance encountered. The variance was then applied in the central limit theorem through the confidence limit approach in equation (1) to derive the number of samples required.

$$n = \frac{(\sigma \; Z\alpha/2)}{d^2} \qquad (1)$$

where

n = number of samples required
σ = standard deviation
$Z_{\alpha/2}$ is the two-tailed standardized normal statistic for probability of 0.95
d = amount of difference allowed between the estimate and the true parameter - 10% of the average found in trial runs [5]

The combination of factors which created the greatest variance dictated the number of runs for all subsequent tests.

[5]Ten percent was picked as a good amount for this study since initial runs showed that ten percent would be sufficient to distinguish significant differences in idle CPU time. One percent or five percent could have been specified, but this level of precision was not deemed necessary.

Conclusion

Many similarities were seen in the structure of the JOVIAL DAIS and Ada executive software. Differences in the two stem from the run-time system in use. The overhead of each run-time system was compared by measuring idle CPU time in all combinations of factors under consideration. These measurements were analyzed statistically. Results are presented in the next chapter.

V. Experimental Results

All factors were evaluated with the exception of the comparison of the compiler effect on Ada executives. All four of the Ada compilers tested had "bugs." As a result of these problems, only one compiler successfully provided executable code. This chapter will begin by describing the problems encountered with each compiler and then proceed to actual test results and statistical interpretation of these results.

Compiler Problems

The compilers used will not be identified by name, but will instead be referred to as A, B, C, and D to avoid mentioning proprietary information. Compiler A was able to run all tests successfully. The other compilers were unable to execute the tests for various reasons. Compiler B compiled and linked successfully, but produced a tasking error during run-time. Compiler C compiled successfully, but encountered an internal linking error. Finally, Compiler D encountered an internal compiler error and could not compile the Task Scheduler package. All compiler problems were submitted to the compiler vendors for debugging if the vendors had a current contract with the Air Force for that compiler.

Even if all four compilers did work, only two would have handled the interrupt version because not all Ada Language Reference Manual chapter 13 features are implemented by each

compiler. The compiler specifications with respect to each pertinent feature are shown in Table VII.

Table VII. Compiler Features

Compiler	Used Successfully	Pragma Interface	Address Clause	Address Clause for Interrupt Handling
A	Yes	Yes	Yes	Yes
B	No	No	No	No
C	No	Yes	No	No
D	No	Yes	Yes	Yes

Compiler A.

Although Compiler A produced useful code, some compiler peculiarities were discovered. The first problem encountered was that of inputting items of type duration interactively from the terminal keyboard. This technique worked in some cases, but failed in others with no apparent reason for the failure. This would have provided a convenient means of entering the minor cycle duration for each test, but an alternate means of loading durations to be tested in an array was used.

The most puzzling problem encountered was found with the timing of the delay statement and assembly coded interrupt procedure. To test the accuracy of the delay versus interrupt minor cycle durations, a test was performed in which the minor cycle duration was set to 0.0312 seconds. This yields a four second major frame. Not surprisingly, since delay is a minimum time period, the delay version took 5.01 seconds for one major frame. As expected, the interrupt version was more precise with a major frame duration of 4.05

seconds.

A discrepancy occurred, however, when the minor cycle duration was increased to 0.5 seconds. With this duration, one major frame should take 64 seconds. With the delay version, however, one major frame took 60.07 seconds. The interrupt version also displayed a similar discrepancy with a major frame time of 60.06 seconds. These results were reported to the compiler vendor for investigation. As of this time, the vendor has not replied with an explanation or modification to correct the problem. Since the long delays which caused the discrepancy were used to investigate delay accuracy and were not used in the actual experiments, the experimental results remain sound.

Compiler B.

Compiler B worked for simple tasking programs where tasks executed sequentially without interleaving. When more complex programs were run with code from this compiler, the program was unable to progress through one major frame before ending with a tasking error statement. Considerable time was spent trying to modify the code to run from this compiler since this was the first Ada to 1750A compiler used. Unfortunately, no solution was found. The compiler vendor was already aware of this problem and may fix it with a later release.

Compiler C.

This compiler was not tested further once the initial linking failure was discovered. Discussions with other users

of this compiler indicated that linking errors were common, and extensive debugging of what the compiler could and could not handle would be futile. This compiler is not currently supported by Air Force contract.

Compiler D.

The compiler error encountered did not give any indication of what could be causing the problem. The Task Scheduler which would not compile was the lowest level package which contains task entry calls. The compiler error is most likely associated with task synchronization and communication. The problem was reported to the vendor, but no solution was offered.

Test Results

The results are reported for each combination of factors in Table VIII. Equation (1) was used to determine how many sample runs were required. Due to the small amount of variance between runs, the five initial runs were sufficient to obtain an answer within 10% of the mean for all test combinations. Therefore, no more runs were necessary.

The results indicate that the Ada model with interrupts has fewer increments than the DAIS or the Ada model with delays. From the overhead measurement section in Chapter IV the number of increments is shown to correspond with idle CPU time. A model which has more idle CPU time is thus more efficient in terms of executive overhead than its counterpart. The results can therefore be interpreted to

show that the Ada model with interrupts was less efficient than the DAIS or the Ada model with delays. There is a significant difference between the Ada version with delays and the DAIS, but this difference is compounded by the

Table VIII. Idle CPU Time for all Combinations of Factors[6]

Run Number	Ada with Delays				Ada with Interrupts			
	Cyclic		Mixture		Cyclic		Mixture	
	Low	High	Low	High	Low	High	Low	High
1	238603	138497	215931	112478	180606	76403	156754	52473
2	238634	138528	217247	114586	180597	76399	158223	54072
3	238700	138560	216593	113972	180597	76407	157481	53294
4	238815	138355	215496	113245	180598	76401	156736	52536
5	238915	138653	215989	113501	180602	76400	157222	53018
Mean	238733	138518	216251	113556	180600	76402	157283	53078
Std Dev	130	108	680	790	4	3	613	651

DAIS

Run Number	Cyclic		Mixture	
	Low	High	Low	High
1	248680	120550	242489	114328
2	248684	120560	242702	114530
3	248693	120545	242564	114428
4	248691	120546	242434	114297
5	248673	120550	242408	114291
Mean	248684	120550	242519	114374
Std Dev	8	6	118	103

[6] Test results given in number of increments per major frame.

Heading codes: Cyclic/Mixture indicates purely cyclic tasks or mixture of cyclic and asynchronous
High/Low indicates high or low workload

interaction of both workload and task mixture. The results are sufficient to provide an equation which can predict idle CPU time, but do not indicate that Ada tasking with delays is better or worse than the DAIS in all cases.

The results from Table VIII shows that the DAIS has more idle CPU time under low workload conditions than Ada tasking with delays regardless of the application task mixture. Ada tasking with delays, on the other hand, performs better than DAIS under high workload with cyclical tasking. Neither version shows a significant advantage under high workload conditions with a mixture of cyclical and asynchronous tasks.

The interrupt driven model displays approximately 30 to 50 percent more processing overhead than the Ada model with delays for all combinations of factors. For example, with cyclical tasks and low workload, Ada with interrupts has 180,600 increments per major frame and Ada with delays has 238,733. This shows the interrupt model having 24 percent greater overhead than the delay model under these conditions.

The results were then analyzed with Analysis of Variance (ANOVA) procedures to determine which factors significantly influenced executive overhead. The null hypothesis for each test was that the factor was not a significant influence; the alternate hypothesis was that the factor provided a significant influence. Initially, the Ada with delay and Ada with interrupt versions were compared with one another. Each factor under consideration was found to have a significant

influence. Since the Ada version with delay statements performed better than the interrupt version, the DAIS was compared with the delay version. Once again, each factor under consideration was found to have a significant influence. To analyze the relative influence of each factor, a stepwise regression analysis was then performed with SAS. The outcome of this analysis indicated which factors should be included in the model and the relative influence of each factor as shown in equation (2).

$$-4,767L - 14946M + 57399W + 16809 \qquad (2)$$

where

Variable	Denotes	Meaning
L		Language - 1-Ada or 0-JOVIAL DAIS
M		Mixture - 0-All cyclical or 1-mixture
W		Workload - 2-High or 4-Low

Conclusion

The effect on idle CPU time was measured and recorded for each factor under consideration except the Ada compiler influence. The compiler influence could not be investigated because only one of the four compilers used was able to produce code which would execute on the 1750A.

All of the remaining factors contributed to the amount of idle CPU time. However, much of the change in idle CPU time resulted from the change in workload through adding asynchronous tasks or decreasing the minor cycle duration. Conclusions will be drawn in the next chapter as to the implications of executive overhead on the applicability of Ada tasking for avionics executives.1

VI. Conclusions

The Ada executive models developed and analyzed in this thesis show that Ada tasking does not posses inherently high overhead when compared with the DAIS. On the contrary, executives can be developed, using Ada tasking with delay statements, which have comparable overhead to the DAIS. These executives, however, are prone to timing problems which may negate Ada tasking utility.

The conclusions drawn in this chapter are limited in the sense that they are based on a comparison of the DAIS and an Ada tasking executive rather than absolute overhead requirements for any avionics executive. Nevertheless, since the DAIS is representative of avionics executives and the application tasks were chosen to model typical avionics systems, the applicability of Ada tasking for avionics executives can be inferred from the results herein. The final analysis of Ada tasking can only be made by using tasking in a complete executive which is ultimately flown and flight tested. The results in this thesis indicate there is a significant amount of risk and problems that may be encountered if Ada tasking is mandated for such a program.

Other limitations of this study include the fact that various architectures including multiprocessor implementations were not evaluated. In addition, several other Ada compilers targeted toward the 1750A exist, but were

not available for use. Likewise, various JOVIAL executives exist, but were not available for experimentation due to proprietary reasons. These limitations must be borne in mind when considering the conclusions reached in this chapter.

The results indicate that Ada tasking with delays has nearly the same overhead as the DAIS and is therefore worthwhile for avionics applications. The problem of cumulative drift, however, confounds this conclusion. Ada delay statements allow cumulative drift because delays are by definition a minimum time interval. Ada is therefore able to perform better under high workloads by postponing the delay expiration.

Although unproven, it is doubtful that the Ada tasking version with delays would be able to handle avionics tasks effectively. Timing problems would most likely have to be handled. Handling these problems implies creating more system overhead as in the interrupt driven model.

While providing more accurate timing, the interrupt driven model is likely to cause excessive demands on the processor. Given the size and weight constraints of embedded avionics systems, increasing processor capacity to accommodate Ada tasking is not a viable alternative.

The DAIS also maintains precise timing by associating the minor cycle duration directly with a system timer. This is nearly analogous to the Ada executive with interrupts. The DAIS therefore limits drift and associated timing problems.

Compiler development and enhancement should make Ada

tasking more efficient. Ada compilers targeted toward the 1750A are relatively immature at this time. As a point of interest, the vendor for compiler A delivered two compiler upgrades during the time this thesis was performed. One of these upgrades made a significant impact since it allowed address clauses to be associated with task entries. Until this upgrade was received, the Ada tasking with interrupt model would not run.

Recommendations

Compiler evolution alone may not be sufficient to make Ada tasking effective for avionics systems. Tasking itself should be redesigned to efficiently handle cyclical tasks. A software interrupt is necessary to provide precise timing for cyclical tasks, but this interrupt need not be tied to a task entry. Rendezvous with an interrupt, as seen in the results, creates a high degree of system overhead.

A table driven approach, such as the DAIS, is one means of providing efficient tasking. Another approach is to develop a pragma "cyclical executive" as proposed by Phillips and Stevenson (1984). This pragma would associate an interrupt with a task entry for each desired frequency. Although this approach promises more accurate system timing, the affect on system overhead could be detrimental as in the Ada tasking with interrupt model developed in this thesis.

A combination of the above proposals shows great promise for providing accurate timing while maintaining low overhead.

The pragma "cyclical executive" can be used to provide system timing, and cyclical tasks can table driven as in the DAIS. In this manner, the task scheduler need not rendezvous with each application task to be run. Instead, the application task can be flagged to run without incurring the overhead of an additional rendezvous.

Future efforts should be made to provide precise cyclical tasking without excessive overhead. Specifically, the overhead associated with table driven cyclical tasks should be investigated. The overhead measurement techniques used in this thesis can be applied to measure idle CPU time and thus compare system overhead of future developments.

Bibliography

Adams, Steven E. and Brian Clausing. "Distributed Avionics Processing Using Ada," IEEE National Aerospace Electronics Conference (NAECON), 2: 979-983 (1983).

Ada Run-Time Environment Working Group (ARTEWG). Catalogue of Ada Runtime Implementation Dependencies. Association for Computing Machinery, November 1986.

Aeronautical Systems Division (ASD): Systems Avionics Division. Computer Program Design Specification for DAIS Mission Software Executive. SA*10122001. 1 January 1980.

Aeronautical Radio Inc. (ARINC), Circulation of Draft 1 of Project Paper 613, "Guidance for Using Ada in Avionic Design." Airlines Electronic Engineering Committtee Letter 86-165/SAI-289, 4 December 1986.

Auerheimer, Brent and Richard A. Kemmerer. "RT-ASLAN: A Specification Language for Real-Time Systems," IEEE Transactions on Software Engineering, SE-12(9): 879-889, September 1986.

Baker, T.P. and G.A. Riccardi. "Ada Tasking: From Semantics to Efficient Implementation," IEEE Software, 2(2): 34-46 (March 1985).

Booch, Grady. Software Engineering with Ada. Menlo Park California: The Benjamin/Cummings Publishing Company, 1983.

Booch, Grady. Software Engineering with Ada (Second Edition). Menlo Park California: The Benjamin/Cummings Publishing Company, 1987.

Burger, Thomas M. and Kjell W. Nielsen. "An Assessment of the Overhead Associated with Tasking Facilities and Task Paradigms in Ada," Ada Letters, 7(1): 1-49-1-58(January, February 1987).

Carrington, J.C. and others. Real-Time Application of Ada Technical Report. Mitre Corp. Bedford, MA. August 1986 (AD-B105247L).

Clapp, Russel M. and others. "Toward Real-Time Performance Benchmarks for Ada," Communications of the ACM 29: 760-778 (August 1986).

Dewar, Robert. New York University. Ada Tasking; Boon or Bane. Address to Special Interest Group for Ada (SIGAda) meeting. Dayton, OH, 15 July 1987.

Digital Avionics Information System (DAIS) Program Office. System Specification for the Digital Avionics Information System. SA 100 100A. 30 July 1977.

Department of Defense. Computer Programing Lan-guage Policy. DoD Directive 3405.1. Washington: Government Printing Office, 2 April 1987.

Department of Defense. Military Standard: Ada Programming Language-ANSI/MIL-STD-1815A. Washington, D.C., January 1983.

Fisher, Robert. Higher Order Language (HOL) Evaluation in a Tactical missile Environment. Tecnnical Report. Hughes Aircraft Co. Canoga Park CA. January 1980.

Friedman, Frank L. and Paul A. T. Wolfgang. "Choosing Ada Tasking Models for Real-time Systems," Defense Electronics. 168-176 (April 1987).

Hanselman, Phillip, Electronics Engineer. Personal Interview. Air Force Wright Aeronautical Labs Data and Signal Processing Group, Wright-Patterson AFB, OH, 15 April 1987.

Helmbold, David and David Lucknam. "Debugging Ada Tasking Programs," IEEE Software, 2(2):47-57(March 1985).

Kamrad, Michael J. II. "Real Life Considerations of Ada Runtime Organizations for Real-Time Applications," AIAA/IEEE Digital Avionics Systems Conference Proceedings. 472-476 (1984).

King, Capt David. Software Group Lead, Personal Interviews. Aeronautical Systems Division Systems Engineering Avionics Facility, Wright-Patterson AFB OH, May 1987 through November 1987.

Leathrum, J. F. "Design of an Ada Run-time System," IEEE 1984 Ada Applications and Environments Conference, 4-13, 1984.

Lindley, Lawrence M. The Use of Higher Order Language for Tactical Avionics Programming. Technical Report. Naval Avionics Center, Indianapolis, IN, February 1980 (AD-B050116L).

Mundie, David A. and David A. Fisher. "Parallel Processing in Ada," Computer, 19(8): 20-25 (August 1986).

Phillips, Steven P. and Peter R. Stevenson. "The Role of Ada in Real Time Embedded Applications," Ada Letters, 3(4): 4.99-4.111(January, February 1984).

Rubey, Raymond J. "Higher Order Languages for Avionics Software--a Survey, Summary, and Critique," IEEE National Aerospace Electronics Conference (NAECON), 2: 945-951 (1978).

Scarpelli, Alfred J. Ada Test and Evaluation. Technical Report. Air Force Wright Aeronautical Labs, Wright-Patterson AFB OH, May 1980 (AD-A087705).

Schnelker, James et al. Tactical Ada Guidance (TAG). Air Force Armament Laboratory report number AFATL-TR-85-54. Contract F08635-83-C-0349. General Dynamics: Data Systems Division, December 1985 (AD-B098625).

Softech Inc. Ada (trade name) Training Curriculum. Real-Time Systems in Ada 1401 Teacher's Guide. Volume 2. Inc., Waltham MA, 1986 (AD-A166352).

Trainor, W. Lynn and others. Efficiency Comparison of JOVIAL-73/I and AN/AYK-15 Assembly Language. Technical Report. Air Force Wright Aeronautical Labs Wright-Patterson AFB OH, January 1977 (AD-A038053).

Weicker, Reinhold P. "Drystone: A Synthetic Systems Programming Benchmark," Communications of the ACM, 27:1013-1030 (July-December 1984).

Witt, Capt Donald J. Using Ada in the Real-Time Avionics Environment: Issues and Conclusions. MS thesis GCS/MA/85D-6. School of Engineering, Air Force Institute of Technology(AU). Wright-Patterson AFB OH, December 1985.

VITA

Major Roger E. Kontak was born on 19 November 1954 in Toledo, Ohio. He graduated from Southport High School in Indianapolis, Indiana, in 1972 and attended the United States Air Force Academy, from which he received the degree of Bachelor of Science in Economics and Management and a commission in the USAF in June 1976. He attended pilot training and received his wings in August 1977. He then served as a C-130 copilot and aircraft commander in the 61st Tactical Airlift Squadron, Little Rock AFB, Arkansas until June 1979. From there he was transferred to the 17th Tactical Airlift Squadron, Elmendorf AFB, Alaska where he served as an aircraft commander and instructor pilot. Before leaving Alaska, Major Kontak served as the Chief of Plans, Current Operations Division of the 616th Military Airlift Group. From there he moved to the 21st Air Force Operations Center, McGuire AFB, New Jersey in July 1983 and served as an officer controller for one year. He remained at McGuire AFB and transferred to the 1701st Mobility Support Squadron where he served as the Chief of Plans, Southwest Asia Branch until entering the School of Engineering, Air Force Institute of Technology, in May 1986. Major Kontak is married and has 3 children.

Permanent address: c/o Rolland E. Kontak
2403 S. Emerson
Indianapolis, Indiana
46203

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) AFIT/GCS/MA/87D-4	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION School of Engineering 6b. OFFICE SYMBOL (If applicable) AFIT/ENC	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code) Air Force Institute of Technology Wright-Patterson AFB OH 45433-6583	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO

11. TITLE (Include Security Classification)
APPLICABILITY OF ADA TASKING FOR AVIONICS EXECUTIVES (UNCLASSIFIED)

12. PERSONAL AUTHOR(S)
Roger E. Kontak, Maj, USAF

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
MS Thesis	FROM _____ TO _____	1987 December	64

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	
12	05		Ada, Tasking, Avionics Executive, Benchmarking
12	08		

LYNN E. WOLAVER 31 Aug 87
Dean for Research and Professional Development
Air Force Institute of Technology
Wright-Patterson AFB OH 45433-6583

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Thesis Chairman: David A. Umphress, Captain, USAF
Assistant Professor, Department of Mathematics and Computer Science.

The purpose of this study was to evaluate Ada tasking performance and its suitability for avionics schedulers known as executives. This was done by comparing variations of Ada executives written by the author with the existing Digital Avionics Information System written in JOVIAL. The comparisons were made by evaluating the system overhead of each model while running a series of representative application tasks.

The study found that Ada tasking had considerably more overhead than its JOVIAL counterpart in order to maintain precise cyclical timing. Another outcome was that several Ada compilers were unable to produce code which could be run on the MIL-STD-1750A computer. This points to the present immaturity of Ada compilers targeted toward embedded aircraft computers.

This thesis adds support for the need to revise standards and develop compilers as necessary to provide an efficient Run Time System for Ada executives.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL David A. Umphress	22b. TELEPHONE (Include Area Code) (513) 255-3098 22c. OFFICE SYMBOL AFIT/ENC

DD Form 1473, JUN 86 *Previous editions are obsolete.*

END
DATE
FILMD
3-88
DTIC